Rapport du projet ASR
TNT – Agents Mobiles

Rédigé par :

Ines CHAABOUNI et Loïc CHALAYER

Encadré par :

Sébastien LERICHE

Département Informatique de l'INT



Table des matières

I. Introduction

Le projet consiste à étudier une solution de diffusion d'un flux de vidéo TNT (Télévision Numérique Terrestre) depuis un serveur relié à un tuner TNT vers un client potentiellement mobile. En fonction du contexte d'exécution du client (type de connexion, puissance, écran...) une adaptation du signal peut être nécessaire à la source (résolution de l'image, nombre d'images par seconde...). De plus, on souhaite pouvoir déplacer l'application cliente à la demande, pour répondre à un besoin de mobilité de l'utilisateur avec changement de terminal.

Côté serveur on a utilisé le logiciel VLC pour recevoir le flux vidéo, le transformer suivant les besoins et le rediffuser en streaming (type HTTP).

Côté client, nous avons développé une interface de visualisation simple (type Swing) permettant d'afficher un flux vidéo via l'API JavaVLC, de piloter les adaptations du serveur et de déclencher la mobilité.

La partie client/serveur/mobilité s'appuie sur le middleware JavAct.

Afin de nous familiariser avec les logiciels et les protocoles, nous avons effectué dans un premier temps des prototypes simples permettant de tester le fonctionnement individuel des différents éléments. Dans un deuxième temps, nous avons choisi une architecture répondant au besoin exprimé, ainsi que les spécifications logicielles du client et du serveur. Enfin, vous avons réalisé la mise en oeuvre complète du projet.

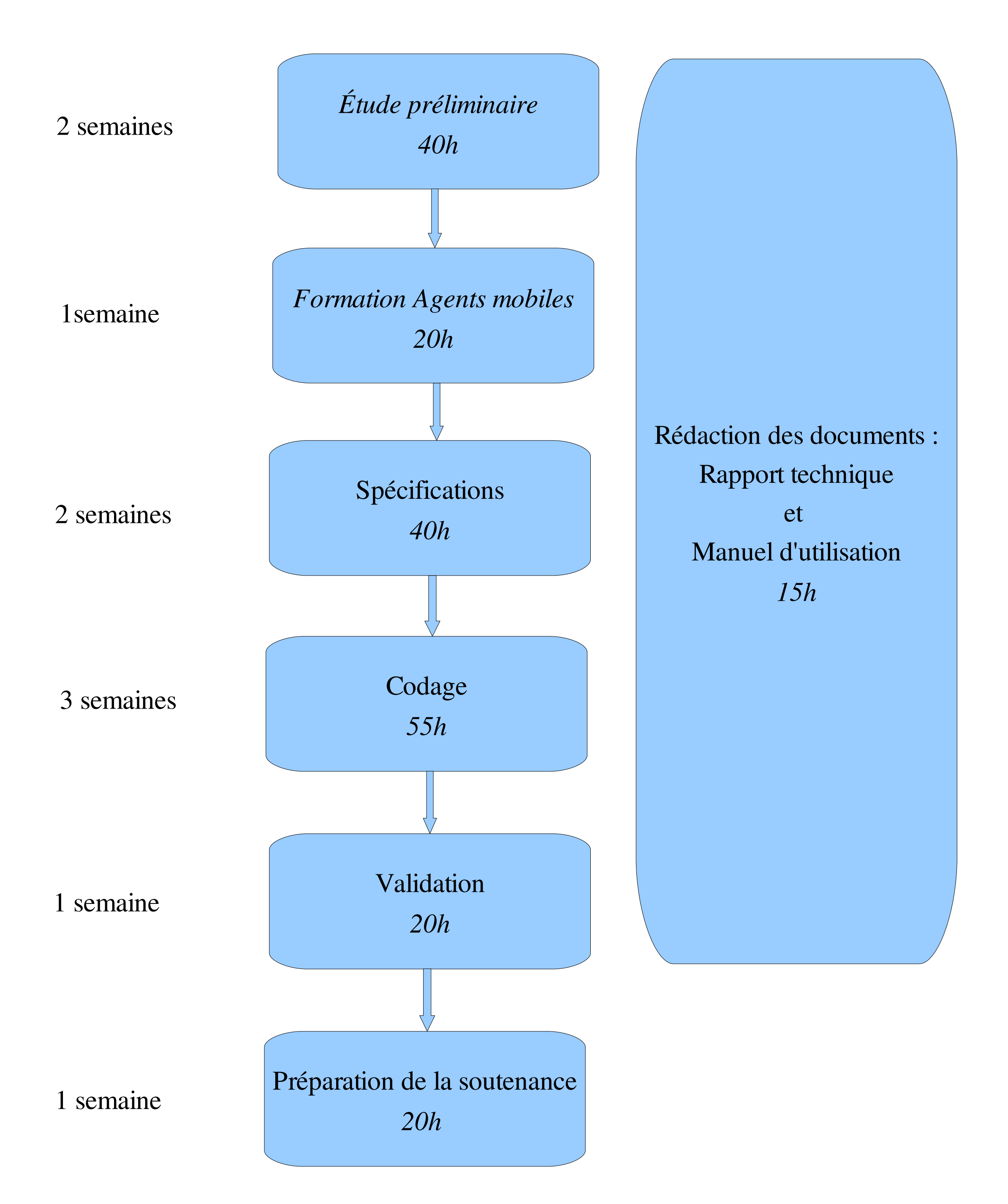
II. Gestion du projet

A. Suivi du projet

On a une réunion hebdomadaire avec l'encadrant pour discuter de l'avancement du projet et et des problèmes rencontrés. Un compte rendu [voir Annexe1] est rédigé après chaque rencontre.

B. Planning prévisionnel et charge horaire

La charge totale est de 190 heures sur 10 semaines, répartie comme suit :



III. Étude préliminaire

A. Mise en place du tuner TNT

Avant de procéder à l'utilisation à proprement parler du tuner TNT dans notre projet, il a fallu se familiariser avec cet élément. Nous avons donc cherché à en connaître les différentes caractéristiques et à découvrir son mode de fonctionnement.

Celui-ci est supporté nativement dans Linux pour les noyaux supérieurs au 2.6. Nous l'avons en l'occurrence testé sur un kernel assez récent 2.6.20-15-generic et un kernel un peu plus ancien le 2.6.22-14-generic.

Voici les messages obtenus au niveau du noyau lors du branchement du tuner TNT:

dmesg

```
[522107.647880] usb 6-2: new high speed USB device using ehci_hcd and address 8
[522107.780765] usb 6-2: configuration #1 chosen from 1 choice
[522107.780974] DVB: registering new adapter (TerraTec/qanu USB2.0 Highspeed DVB-T Receiver).
[522107.781233] input: TerraTec/qanu USB2.0 Highspeed DVB-T Receiver remote control as /class/input/input21
```

Voici la liste des modules chargés dans le noyau lors de l'utilisation du tuner TNT

```
cinergyT2              17540  0
dvb_core               82216  1 cinergyT2
usbcore               138632  13 cdc_ether,usbnet,cdc_acm,cinergyT2,snd_usb_audio,sn
```

lsmod | grep cinergyT2

Voici maintenant la liste des pacages biangle qu'il a fallu rajouter afin de pouvoir utiliser correctement le logiciel sur une distribution Intuber 2.6.20-15-generic:

- libertine-extractions (obligatoire): codes nécessaires à la lecture des vidéos
- dvb-utils (facultatif): permet entre autre de monitorer la puissance du signal, de scanner les fréquences et de générer un fichier channels.conf

Recherche des chaînes:

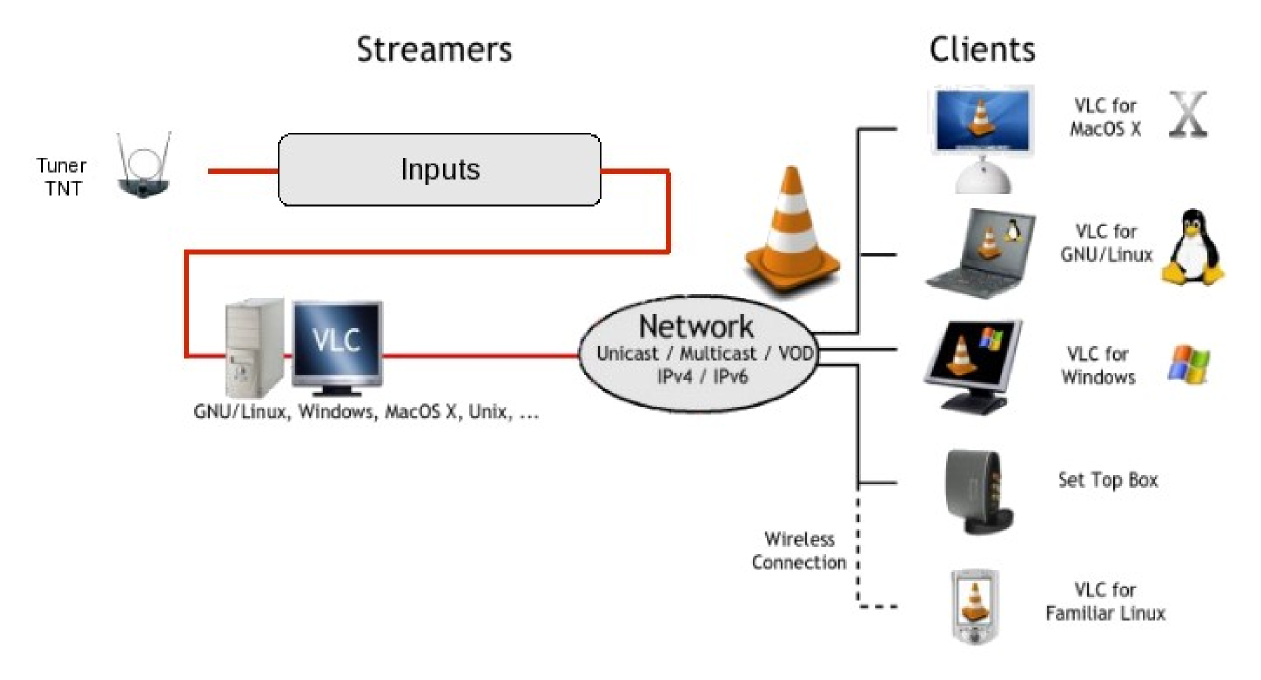
Lors de la création du fichier channels.conf il faut prendre garde à décaler le signal des fréquences de +167Khz. En effet celles-ci ne sont pas détectées convenablement par l'utilitaire scan. Il est ensuite possible de créer une playlist des chaînes au format m3u et de la passer au logiciel vlc.

Ce module auto-alimenté par USB 2.0 est donc très simple d'utilisation et ne nécessite aucune manipulation particulière.

B. Tests du serveur VLC

i. Mise en place

VLC est une solution complète pour la lecture et la diffusion de vidéo par réseau. Il peut être utilisé comme un serveur pour diffuser des fichiers MPEG-1, MPEG-2 et MPEG-4, des DVDs et des flux vidéo réseau en unicast ou multicast. Il peut également être utilisé comme un client pour recevoir, décoder et afficher des flux MPEG sous de nombreux systèmes d'exploitation.



Voici quelques exemples de lignes de commande pour lancer un serveur de diffusion avec vlc:

- vlc -I ncurses --ttl 20 /home/kubuntu/Desktop/vlc_tnt_paris.m3u --sout='#transcode{vcodec=mp4v,acodec=mpga,vb=500,ab=64,scale=0.6}:standard{access=http{user=test,pwd=test},mux=ts,url=:1234}' 2> /dev/null
- vlc -I ncurses --ttl 20 /home/kubuntu/Desktop/vlc_tnt_paris.m3u --sout='#transcode{vcodec=mp4v,acodec=mpga,vb=500,ab=64,scale=0.6}:standard{access=udp,mux=ts,url=10.0.100.254}' 2> /dev/null

Voici la liste des modules que nous avons utilisés:

- standard permet d'envoyer le flux via un module de sortie : par exemple, UDP, fichier, HTTP...

- transcode est utilisé pour transcoder (décoder et ré-encoder le flux en utilisant un codec et/ou un débit différent) l'audio et la vidéo du flux d'entrée. Si les modules d'entrée ou de sortie ne permettent pas le contrôle du débit (réseau, périphériques d'acquisition), ceci est fait "à la volée", en temps réel. Ceci peut nécessiter un charge importante en CPU, en fonction des paramètres choisis. Les autres flux, tels que fichiers et disques, sont transcodés aussi rapidement que le permet le système.

Module Standard: ce module enregistre le flux dans un fichier ou le diffuse sur le réseau, après l'avoir multiplexé. Les options disponibles et que nous utilisons sont :

access=

Ces options permettent de sélectionner le medium utilisé pour enregistrer ou diffuser le flux. Cette option est obligatoirement requise. Les choix possibles sont :

udp : diffuse vers une adresse UPD unicast ou multicast.Les options sont : caching=<durée en ms> pour définir le temps pendant lequel VLC devrait garder les données en tampon avant de les envoyer, ttl=<ttl> pour définir le ttl des paquets UDP envoyés, group=<nombre de paquets> pour envoyer les paquets par rafales plutôt que un par un, late=<durée en ms> pour éliminer les paquets qui arrivent trop en retard à cette étape de la chaîne, raw si vous ne voulez pas attendre que le MTU soit rempli avant d'envoyer les paquets.

http : diffuse par HTTP. Les options sont : user=<nom d'utilisateur> pour activer l'authentification basique par HTTP et définir l'utilisateur, pwd=<mot de passe> pour définir le mot de passe mime=<type mime> pour définir le type MIME retourné par le serveur.

rtp : diffuse par RTP. Notez que c'est un module assez vieux. Il ne permet pas le RTSP et peux uniquement être utilisé pour diffuser des flux TS. Veuillez regarder le module de sortie rtp pour un support complet du RTP.Les options sont les mêmes que pour pour l'udp.

mux=

Cette option vous permet de choisir la méthode d'encapsulation utilisée dans le flux. Cette option est obligatoire. Les options disponibles sont :

ts : le muxer MPEG2/TS. C'est le muxer standard utilisé pour diffuser du MPEG 2. Ce muxer peut être utilisé avec n'importe quelle méthode d'accès. Les codecs supportés sont MPEG 1/2/4, MJPEG, H263, H264, I263, WMV 1/2 et theora pour la vidéo, l'audio MPEG, AAC et a52 pour le flux audio.Les options sont : pid-video=<pid> pour choisir le PID de la piste vidéo, pid-audio=<pid> pour choisir le PID de la piste audio, pid-spu=<pid> pour choisir le PID de la piste de sous-titres, pid-pmt=<pid> pour choisir le PID de la PMT (Program Map Table), tsid=<id> pour choisir l'ID du flux TS, shaping=<delai de formation en ms > pour choisir l'intervalle minimum durant lequel le débit du flux doit rester constant, pour les flux à débit variable, use-keykey-frames utiliser les images I comme limites de délai de formation, pcr=<intervalle PCR en ms> permet de choisir à quel intervalle le PCR (Program Clock References) doit êtreenvoyé, dts-delay=<delai en ms> permet de retarder le PTS (Presentation Time Stamps) du DTS (Decoding Time Stamp) du delai indiqué, crypt-audio permet d'activer le chiffrement de la piste audio en utilisant l'algorithme CSA, csa-ck=<clef sous forme d'un mot de 16 caractères> permet de choisir la clef à utiliser pour le chiffrement CSA.

url=

Cette option permet de donner les informations sur l'emplacement où le flux doit être enregistré ou envoyé.

Mode transcode: Vous pouvez utiliser ce module pour transcoder un flux, c'est-à-dire changer ses codecs ou ses débits d'encodage. Quelques ajustements supplémentaires peuvent être réalisés pendant cette phase, comme changer la taille, désentrelacer, rééchantillonner, etc.

Note: Selon le débit original du flux et les options choisies, transcoder peut être une tache très intensive pour le processeur. En conséquence, la diffusion du flux transcodé en temps réel peut mener à des pertes de frame ou une mauvaise image dans certainscas, quand il manque des ressources.

Les options disponibles sont :

vcodec=

Cette option permet de choisir le codec des pistes vidéos dans lequel le flux d'entrée doit être transcodé: MPEG-1 video , MPEG-2 video , MPEG-4 video , DivX 1/2/3 video , WMV 1/2 , H/I 263 , MJPEG , Theora

vb=

Cette option permet de régler le débit flux video transcodé en kbit/s

scale=

Cette option permet de donner le ratio dans lequel doit être redimensionnée la vidéo pendant le transcodage. Cette option peut être particulièrement utile pour aider à réduire le débit d'un flux.

width=

Cette option permet de donner la largeur de la vidéo transcodée en pixels.

height=

Cette option permet de donner la hauteur de la vidéo transcodée en pixels.

acodec=

Cette option permet de spécifier le codec dans lequel la piste audio doit être transcodée: MPEG Layer 2 audio , MPEG Layer 3 audio , AC3 , MPEG-4 audio , Vorbis/Speex , FLAC , PCM , μ-law/A-law

ab=

Cette option permet de fixer le débit du flux audio transcodé en kbit/s.

ii. Résultats des tests

a) Influence du muxer sur le débit en Mode Standard

http	ts (*) (refresh lent)	ps (file ou http)	mpeg1 (file ou http)	ogg (file ou http)	asf (file ou http) (refresh lent)	asfh (mmsh)	avi (file)	mpjpeg (http)
Débit d'entrée (kb/s)	4753 (udp:4423)	4213	4383	4405	4369			do not work
Débit du flux (kb/s)	4325 (udp:4483)	4059	4167	4442	4214			do not work

Remarque: Quelques pertes d'images et de trames audio en udp et http

On utilise le muxer ts pour effectuer les tests (nombreux codecs supportés).

b) Influence du codec sur le débit et la qualité

udp+ts	mp1v	mp2v	mp4v	DIV2	DIV3	WMV2	h264
Débit d'entrée (kb/s)	911	993	989	954	774	877	660
Débit du flux (kb/s)	895	869	865	822	545	842	537
Qualité Image	***	***	***	**	*	**/***	****

Remarque: Pour le codec h264 , les scènes d'actions paraissent lentes. Consommation importante de ressources d'autre part.
En outre si on ne précise pas la taille de l'image, elle varie selon les codecs employés.

Influence des arguments ab et vb: Ils permettent de régler le débit du flux audio et vidéo à une limite

à ne pas dépasser.
Influence du champ scale: Ce champ permet seulement de modifier la taille de l'image et donc d'adapter l'image à un bas débit en réduisant sa taille mais ne permet pas de diminuer le flux.

Remarque Générale : Ces débits sont donnés à titre indicatif, ils évoluent fortement en fonction du type d'émission.

C. Formation agents mobiles

i. Présentation de la programmation par agents mobiles

« La programmation par agents mobiles est un paradigme de programmation des applications réparties, susceptible de compléter ou de se substituer à d'autres paradigmes plus classiques tel le passage de messages, l'appel de procédure à distance, l'invocation d'objet à distance, l'évaluation à distance. Elle est d'un grand intérêt pour la mise en œuvre d'applications dont les performances varient en fonction de la disponibilité et de la qualité des services et des ressources, ainsi que du volume des données déplacées. Le concept d'agent mobile facilite en effet la mise en œuvre d'applications dynamiquement adaptables, et il offre un cadre générique pour le développement des applications réparties sur des réseaux de grande taille qui recouvrent des domaines multiples.

...

Un agent logiciel est une entité autonome capable de communiquer, disposant d'une connaissance partielle de ce qui l'entoure et d'un comportement privés, ainsi que d'une capacité d'exécution propre. Un agent agit pour le compte d'un tiers (un autre agent, un utilisateur) qu'il représente sans être obligatoirement connecté à celui-ci, réagit et interagit avec d'autres agents.

Un agent mobile peut se déplacer d'un site à un autre en cours d'exécution pour accéder à des données ou à des ressources. Il se déplace avec son code et ses données propres, mais aussi avec son état d'exécution. L'agent décide lui-même de manière autonome de ses mouvements. Ainsi, la mobilité est contrôlée par l'application elle-même, et non par le système d'exécution comme dans le cas de la migration de processus dans les systèmes opératoires.

En pratique, la mobilité d'agent permet de rapprocher client et serveur et en conséquence de réduire le nombre et le volume des interactions distantes (en les remplaçant par des interactions locales), de spécialiser des serveurs distants ou de déporter la charge de calcul d'un site à un autre. Une application construite à base d'agents mobiles peut se redéployer dynamiquement suivant un plan pré-établi ou en réaction à une situation particulière, afin par exemple d'améliorer la performance ou de satisfaire la tolérance aux pannes, de réduire le trafic sur le réseau, ou de suivre un composant matériel mobile. La mobilité du code offre un premier niveau de flexibilité aux applications. La décentralisation de la connaissance et du contrôle à travers les agents, et la proximité physique entre les agents et les ressources du système renforce la réactivité et les capacités d'adaptation.

La mobilité ne se substitue pas aux capacités de communication des agents (la communication distante reste possible) mais les complète ; afin de satisfaire aux contraintes des réseaux de grande taille ou sans fil (latence, non permanence des liens de communication), les agents communiquent par

messages asynchrones.»[1]

ii. JavAct : un intergiciel Java pour les agents mobiles adaptables

« JAVACT est une bibliothèque Java pour la programmation d'applications concurrentes, réparties et mobiles à base d'agents. Actuellement en cours de développement à l'IRIT, elle est distribuée sous forme de logiciel libre sous licence LGPL.

La plate-forme JAVACT s'appuie sur les concepts d'acteur et d'implémentation ouverte, et permet une programmation de haut niveau en Java standard en faisant abstraction des mécanismes de bas niveau (processus légers, synchronisation, RMI, Corba, ...). JAVACT a été conçue afin d'être minimale et maintenable à moindre frais, portable, et exploitable par un programmeur Java "moyen" initié aux acteurs. Dans sa version actuelle (0.5.1), JAVACT s'appuie sur le SDK 1.4 et RMI ; il n'y a pas de préprocesseur ni de modification de la machine virtuelle ce qui permet d'utiliser tous les outils standard de l'environnement Java.

Les acteurs sont des objets actifs qui communiquent de manière asynchrone, et dont le comportement peut changer en cours d'exécution (interface non uniforme). Les acteurs sont des entités autonomes naturellement mobiles, et la mobilité n'induit pas de modification sémantique. JAVACT offre des mécanismes pour la création d'acteur, leur changement d'interface, leur répartition et leur mobilité, leur adaptation statique et dynamique, les communications (locales ou distantes). La mobilité et l'adaptation sont effectives au moment du changement de comportement ; ainsi, on ne diminue pas l'expressivité et le niveau abstraction,et on contourne les inconvénients de la mobilité faible de Java.

Une application JAVACT s'exécute sur un domaine constitué par un ensemble de places qui peut évoluer dynamiquement. Une place est une machine virtuelle Java (au besoin, on peut simuler la répartition en créant plusieurs places sur le même site physique). Le même programme JAVACT est exploitable indifféremment dans un environnement réparti ou non. L'allocation de ressources aux activités concurrentes est (dans la version actuelle)laissée à la charge de la machine virtuelle Java. »[2]

iii. Déploiement d'un agent mobile JavAct sous Eclipse

Les étapes de déploiement sont les suivantes :

- Créer un nouveau projet JavAct.
- Ajouter une interface, qui hérite de *BehaviorProfile* et de *ActorProfile* (pour spécifier que l'interface représente un comportement d'acteur et également un profil d'acteur).
- Dans cette interface, insérer les méthodes correspondant aux messages reçus par l'acteur.
- Lancer la compilation de l'interface en cliquant sur *Project* puis *Build Project*.
- Dans le *Package Explorer* d'Eclipse, sélectionner l'interface pour activer les fonctionnalités spécifiques du plugin JavAct. Cliquer sur l'icône *JavActGen et* vérifier qu'il n'ya pas d'erreurs.
- Une fois les nouvelles classes générées, compléter le squelette . Attention, le programme principal n'est pas lui-même un acteur.

1 http://fr.wikipedia.org/wiki/Agent_mobile

2 http://www.irit.fr/PERSONNEL/SMAC/arcangeli/JavAct_fr.html

- Lancer le projet via les commandes appropriées d'Eclipse.

iv. Applications

Nous présentons ici deux applications que nous avons faites pour mieux assimiler la notion d'agents mobiles et pratiquer la programmation avec JavAct.

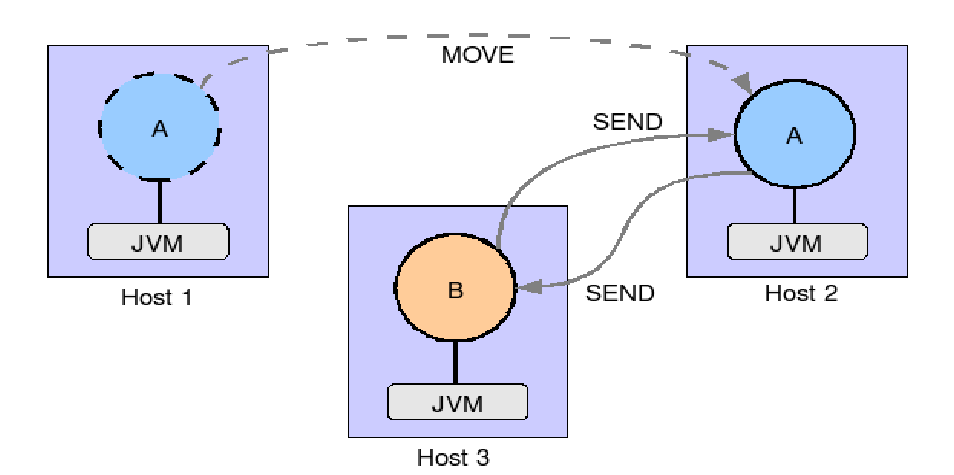
a) *Crible d'Eratosthène*

Il s'agit de déterminer les n premiers nombres premiers. Pour cela nous utilisons la capacité des agents à changer de comportement.

Nous utilisons un acteur par nombre premier, qui peut avoir 2 comportements. Si ce n'est pas un acteur terminal, pour chaque message reçu (contenant un nombre à cribler) s'il le divise alors il le jette, sinon on le passe à l'acteur suivant (prochain nombre premier). Si c'est le dernier du pipeline, pour chaque message reçu s'il le divise alors il le jette, sinon il crée un nouvel acteur terminal avec cette valeur (qui affiche sa valeur), et devient un acteur non terminal dont le suivant est l'acteur qu'il vient de créer.

Le programme principal qui prend une valeur n en entrée, crée le premier acteur terminal (2) et lui transmet la série de nombres à cribler (de 3 à n).

b) *Messagerie*



Il s'agit de réaliser un prototype d'une application de messagerie instantanée ayant la capacité de se déplacer au gré de l'utilisateur. Nous utilisons alors la capacité des agents à se déplacer d'un site à un autre pour réaliser la mobilité sans avoir besoin d'un serveur tiers. Nous utilisons aussi leur capacité à s'échanger des messages de façon asynchrone. De plus chaque agent interagit avec une IHM. Nous avons besoin de ces aspects pour notre projet.

D. Java Media Framework (JMF)

i. Présentation de JMF

JMF, pour Java Media Framework, est une API permettant d'incorporer des données de type audio ou vidéo dans des applications Java. Celle-ci nous a permis dans le cadre de notre projet de lire un flux vidéo en provenance d'un serveur de diffusion VLC et d'afficher ce flux sur un client compatible JAVA.

Voici une liste non exhaustive des diverses fonctionnalités proposées par JMF:

- Lire différents types de fichiers multimédias dans une application **Java** ou un applet. Voici les formats supportés et disponible sur le site de sun (http://java.sun.com/products/java-media/jmf/2.1.1/formats.html) : ***AIFF, AVI, GSM, HotMedia, MIDI, MPEG-1 Video, MPEG Layer II Audio, QuickTime, Sun Audio, Wave.***
- Capturer des données audio et vidéo depuis une caméra ou une webcam puis les enregistrer dans différents formats. Voici une liste d'équipements fonctionnels avec JMF 2.1.1 : ***JavaSound (16-bit, 44100, 22050, 11025Hz, 8000Hz linear), SunVideo, SunVideoPlus, VFW, Intel Create & Share, Diamond Supra Video Kit, QuickCam VC (camera), e-cam (camera), Winnow Videum, Creative Web Cam II, Miro Video DC30, Iomega Buz, QuickCam Home USB (Camera), Smart Video Recorder III.***
- Transmettre des flux audio et vidéo en temps réel sur des réseaux
- Lire des médias temps réel depuis une source réseau
- Diffuser simultanément à de nombreux clients de la vidéo ou de l'audio (broadcast)

JMF est donc une API permettant l'exploitation du streaming avec le langage java. Elle offre de nombreuses possibilités et est très simple d'utilisation. Elle a été conçu dans sa version 2 pour répondre aux attentes suivantes :

- Permettre la capture de données multimédias
- Permettre le développement d'application java utilisant le streaming ou les conférences vidéos.
- Permettre l'accès à un large type de données
- Offrir un support pour le protocole RTP (Real-Time Transport Protocol)

JMF se décompose en deux modules distinct :

- L'API de base : Elle fournit toute une architecture permettant de gérer l'acquisition, le traitement et l'affichage de données multimédias. On peut alors facilement à, l'aide de JMF, créer une applet ou une application qui présente capture, manipule ou enregistre des flux multimédia. On trouve alors différents outils comme les Players qui vont permettre la visualisation et le traitement des données. On pourra alors grâce à eux traiter le flux vidéo et permettre les options que l'on souhaite sur le lecteur media.

- L'API RTP : Jusque là, JMF ne permettait que de lire, traiter et présenter un flux arrivant à un utilisateur. Grâce à l'API RTP on va maintenant pouvoir transmettre un flux et ainsi créer sn propre serveur de streaming. On peut maintenant capturer un flux à partir d'une caméra ou un micro et le transmettre à différents utilisateurs ou encore centraliser un ensemble vidéos et sons et les transmettre sur demande.

ii. Intégration de JMF à l'environnement de travail

Dans la mesure où nous avons utilisé seulement une petite partie des fonctionnalités de JMF (c-à-d lecture d'un flux vidéo), il n'y a eu aucune incompatibilité matérielle (c-à-d pas de problème lié à une source de capture par exemple).

iii. Installation de JMF

Les kits JMF sont disponibles et accessibles à l'adresse suivante http://java.sun.com/products/java-media/jmf/2.1.1/download.html. Nous avons dans un premier temps voulu utilisé le pack recommandé pour Linux. En effet nos machines de développement étaient celles de la salle ASR qui tournent sous Fedora Core 6 avec un noyau 2.6.18-1. Cependant il nous a été impossible de réaliser l'extraction du fichier binaire associer à cette version **jmf-2_1_1e-linux-i586.bin**. En effet après avoir donné les droits en exécution etlancé le binaire téléchargé, une erreur se produit.

```
# cp jmf-2_1_1e-linux-i586.bin working_directory/
# cd working_directory/
# chmod a+x jmf-2_1_1e-linux-i586.bin
# ./jmf-2_1_1e-linux-i586.bin
```

```
Permit writing local files from an applet? (recommend no, see readme.html) [yes or no]
yes
Unpacking...
tail: Ne peut ouvrir `+309' en lecture: Aucun fichier ou répertoire de ce type
Extracting...
./install.sfx.9007: line 1: ==: Aucun fichier ou répertoire de ce type
./install.sfx.9007: line 3: syntax error near unexpected token `)'
QQR���▸��rqp����"���hj��W�jYn�d!���
```

Après avoir pris connaissance de la licence, le binaire vous propose de permettre à un applet d'enregistrer du son et de la vidéo à partir d'un système de capture local. Ensuite, il vous propose d'autoriser un applet à écrire des fichiers localement sur le disque dur. Après avoir répondu à ces quelques questions, le script échoue. Il est à noter que nous avons essayé d'installer ce package sur d'autre distribution linux et notamment sur une Kubuntu Gutsy avec un noyau 2.6.22-14-generic. Celle-ci s'est déroulée avec succès. Il semble donc que le package Linux JMF est un problème de compatibilité avec les Fedora Core toutes versions confondues.

Pour palier à ce problème nous avons décidé d'installer le kit JMF Cross Platform qui se présente sous la forme d'un fichier Zip **jmf-2_1_1e-alljava.zip**. Il suffit alors d'extraire cette archive dans notre répertoire de travail.

```
# cp jmf-2_1_1e-alljava.zip working_directory/
# cd working_directory/
# unzip  jmf-2_1_1e-alljava.zip
```

Il est nécessaire ici de personnaliser les paramètres de JMF à l'instar de la version binaire du package. On pourra entre autres choisir les périphériques audio ou vidéo ou bien encore activer/désactiver des fonctionnalités multimédia.

Ajouter JMF au CLASSPATH

Pour compléter l'installation il faut ajouter JMF dans le classpath de la machine.

```
# export JAVA_HOME=/asr2007/chalayer/jdk1.6.0_03
# export PATH=$JAVA_HOME/bin:$PATH:.
# export JMFHOME=/asr2007/chalayer/JMF-2.1.1e
# export CLASSPATH=.:$JMFHOME/lb/jmf.jar :$JMFHOME/lib/customizer.jar
```

A tout moment on pourra décider de créer un nouveau jar et de l'inclure dans le classpath. Pour se faire il suffit de lancer le «customizer» fourni par jmf.

```
# cd $JMFHOME/lib
# ./jmfcustomizer
```

Architecture et fonctionnement de JMF

Nous allons maintenant évoquer les différents concepts proposés dans l'API JMF:

- Source de données (**DataSource**)
- Périphérique de capture (**CaptureDevice**)
- Lecteur (**Player**)
- Processeur (**Processor**)
- **Datasink**
- **Format**
- **Manager**

La source de données est un flux de données qui encapsule un média. Elle peut contenir des données audio, vidéo ou même un mélange des 2. Une source de données peut être instanciée à partir d'un fichier local ou en réseau. Elle contient la localisation du média (c-à-d adresse du fichier source par exemple), le protocole et le logiciel utilisé pour délivrer le média. Une source de données peut être envoyée dans un lecteur pour pouvoir être présentée.

Le périphérique de capture comme son nom l'indique est un dispositif matériel capable d'acquérir des données (webcam, micro, caméra ...).

Un lecteur est un objet qui prend en entrée un flux de données (audio et/ou vidéo) et effectue un traitement pour pouvoir présenter ces données. Mais avant de pouvoir présenter un média, un lecteur

doit passer par différentes étapes pour se préparer à la lecture. Voici la liste des différents états :

- Unrealized: Dans cet état, l'objet Player a été instancié mais il ne connaît encore rien du média qu'il aura à lire.
- Realizing: Le lecteur passe en état Realizing quand on appelle la méthode realize(). Le lecteur détermine alors les ressources dont il aura besoin.
- Realized: Le lecteur sait qu'elles sont les ressources nécessaires et détient les informations concernant le média qu'il aura à lire.
- Prefetching: Le lecteur passe en état Prefetching quand on appelle la méthode prefetch(). Le lecteur précharge alors le média, s'assure d'avoir l'exclusivité sur les ressources matérielles.
- Prefetched Le lecteur est prêt à lire le média.
- Started Passage en Started lors de l'appel à la méthode start(). La lecture démarre.

Le processeur est une sorte de lecteur. L'interface Processor implémente d'ailleurs celle du Player. La seule différence est donc qu'il permet d'accéder aux traitements effectués dans le lecteur. Le flux sortant d'un processeur peut être récupéré par un autre processeur, ou par un lecteur. Il possède les 6 états du lecteur et en rajoute 2 : Configuring et Configured. Ces états sont à placer entre les états Unrealized et Realizing.

- Configuring : Le processeur entre en état Configuring quand la méthode configure() est appelée. Un processeur a atteint cet état quand il a accès à la source de données et aux informations concernant le format.
- Configured : Le processeur passe en état Configured quand il s'est connecté à la source de données et qu'il a fini de déterminer le format.

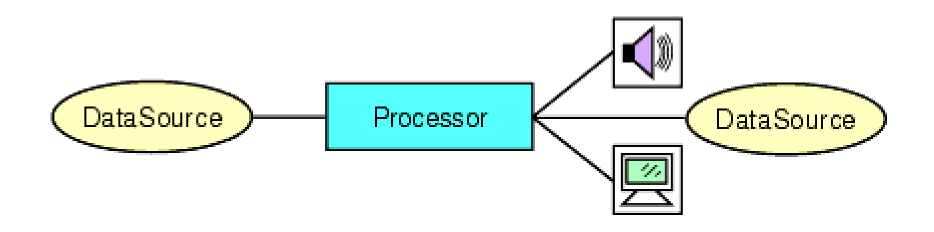
Un Datasink récupère le média d'une source de données et le redirige vers une destination. Un Datasink permet par exemple d'enregistrer un média dans un fichier.

Un objet format contient le format exact d'un media. Il ne contient pas de paramètres spécifiques d'encodage ou d'information sur la durée, il donne seulement le nom du format d'encodage et le type de données nécessaires.

Un manager est un objet qui permet de faire la jonction entre les différents éléments et de simplifier l'utilisation des différentes fonctions de l'API. Il permet par exemple de créer un lecteur à partir d'une source de données. Au final il permet de centraliser les différentes fonctions :

- Manager : Cette classe permet la construction des objets principaux, autrement dit les Players, Processors, DataSources et DataSinks(permettre l'enregistrement d'un flux ou la capture)
- PackageManager : Maintient un registre de packages contenant des classes JMF customisés (Players particuliers...)
- CaptureDeviceManager: Maintient une liste des périphériques de capture valable dans JMF
- PlugInManager : maintient une liste des plugins JMF de traitement utilisables (Multiplexers, Demultiplexers, Codecs, Effects, Renderers)

Schéma de fonctionnement



Intégration de JMF

Notre problématique était donc de récupérer un flux rtp en provenance du serveur vlc et de le faire afficher par le client JMF.

Première étape : Lecture d'un fichier vidéo simple

Pour ce faire nous sommes parti d'un exemple existant et disponible à l'adresse suivante **http://www-igm.univ-mlv.fr/~dr/XPOSE2005/boitel/base.php** et nous l'avons intégré dans éclipse.

```
public static void init {

  //adresse de la source correspondant à un flux RTP
  String SourceAddress = "rtp://192.168.0.1:33335/video/1";

  //création de la source
  MediaLocator SourceLocator = new MediaLocator(SourceAddress);

  //Vérification que la source existe
  if(SourceLocator == null)
  {
    System.out.println("pas de source");
    System.exit(-1);
  }
  else
  {
    System.out.println("Connecté au flux");
  }

  //l'objet Player qui va présenter le flux
  Player player;
  try
  {
    //Creation du player dans l'etat realized avec la source
    player = Manager.createRealizedPlayer(SourceLocator);
    //Demarrage du Player
    player.start();
    //Creation d'une fenêtre de test
    JFrame fenetre = new JFrame("Player");
    fenetre.setSize(180,160);
    //Ajout du Composant visuel du Player dans la fenetre
    fenetre.getContentPane().add(player.getVisualComponent());
    fenetre.setVisible(true);
  }
  catch (Exception e)
```

```
  {
    e.printStackTrace();
  }

}
```

Cet exemple qui montre la création d'un player classique nous a permis de faire afficher une vidéo dans un panneau.

Seconde étape: Lecture d'un flux rtp en provenance de VLC
L'API JMF nous permet de gérer des sessions RTP très simplement. Elle fournit une classe Manager "RTPManager" qui centralise toutes les fonctions liées à une session RTP. Cette classe est donc utile pour la réception de flux RTP.

L'accès à une ressource par un serveur RTP se fait en utilisant une adresse spécifique de la forme rtp://[adresse ip]:[port]/[type]/. On peut alors créer un objet MediaLocator à partir de cette adresse qui sera utilisé comme source du flux RTP. On peut ensuite utiliser l'API de base JMF pour présenter le flux arrivant dans une application ou une applet.

Pour cette étape nous nous sommes en particulier inspiré des exemples présentés sur le site de sun (http://java.sun.com/products/java-media/jmf/2.1.1/solutions/AVReceive3.java). Cependant nous avons été dans l'impossibilité de faire afficher un flux rtp par notre player. Le problème semblait venir du format de notre flux rtp qui n'était pas reconnu par JMF. Pour résoudre ce problème nous avons donc pensé à modifier légèrement notre architecture. L'idée était donc d'utiliser un «proxy rtp» entre le serveur vlc et le client JMF.

Troisième étape: Émission et lecture d'un flux rtp
L'API JMF permet donc également d'envoyer un flux rtp. Dans un premier temps l'objet Processor va nous permettre d'appliquer les traitements nécessaires avant la transmission sur le réseau. On va alors pouvoir paramétrer le type d'encodage, la qualité et tout les paramètres liés au fichier que l'on souhaite transporter. Une fois ces traitements effectués sur le fichier on utilise le PushDatasource qu'il nous renvoie en sortie pour créer les flux d'envoi. Le RTPManager nous permet alors de créer une session RTP pour chaque flux que l'on souhaite transiter.

Là encore nous nous sommes basés sur les exemples de Sun (http://java.sun.com/products/java-media/jmf/2.1.1/solutions/AVTransmit3.java). Nous pensions que si le flux rtp était correctement formé, nous allions pouvoir afficher la vidéo. Cependant nous ne sommes parvenu à faire communiquer les deux entités. L'erreur retournée était celle d'un flux rtp vide. En conséquence, nous avons finalement décidé d'abandonner l'API JMF et de lui préférer la solution JVLC.

E. Java VLC

Java-vlc représente une alternative sérieuse à l'utilisation de l'API JMF. Cette API a pour but d'intégrer le lecteur VLC dans un environnement Java. Celle-ci permet donc à l'instar de VLC de lire de nombreux fichiers vidéos.

Le désavantage majeur de cette solution est la difficulté d'intégration de cette API dans notre environnement. En effet des versions pré-compilés existent pour linux mais seulement pour des architectures 64bits. Il faut donc compiler à la main les sources.

Nous avons cependant utilisé le paquet « java-vlc » disponible dans la distribution Fedora Core8.

Pour pouvoir ensuite utiliser cette bibliothèque sous éclipse il suffit de réaliser les manipulations suivantes:

- copier l'arborescence org/videolan/etc dans le projet eclipse
- copier le fichier libjvlc.so et libmawt.so dans le projet eclipse
- lancer le projet avec l'option "-Djava.library.path=." afin qu'il trouve les librairies

Le fonctionnement de l'API est assez simple. Il consiste à la création d'un playlist. Il faudra ensuite appeler une méthode de lecture sur cette playlist. Les différentes fonctionnalités présentes dans VLC pour modifier la taille de l'image, faire des captures d'écrans, régler le volume de la vidéo se retrouvent facilement utilisable grâce à l'API.

IV. Mise en place de la solution

A. Spécifications

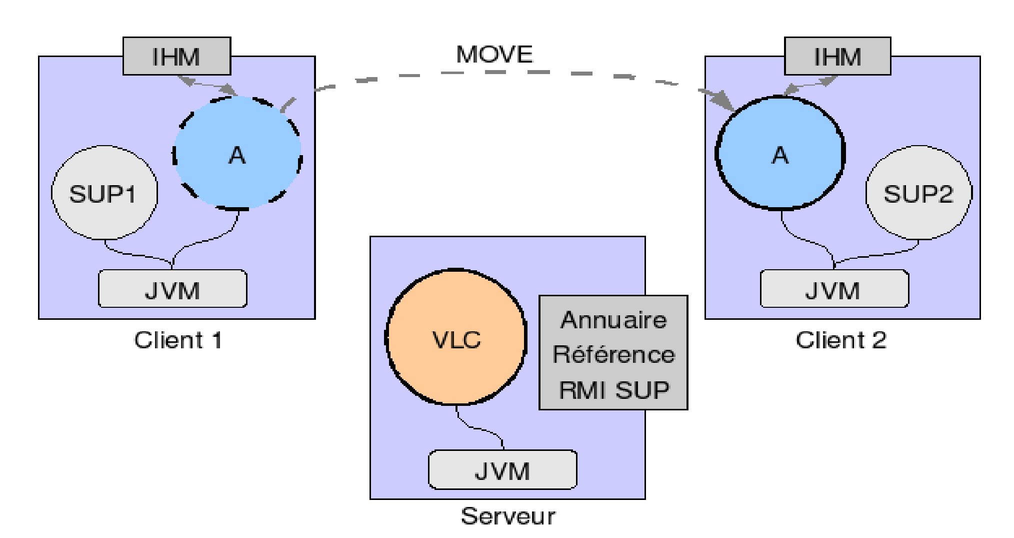
Les cas d'utilisation que nous avons prévus sont les suivants :

- Interactions avec l'utilisateur :
  - Changement de Chaînes
  - Changement de l'adresse IP de client
  - Changement de la taille de l'image, la résolution et la bande passante disponible.
- Interaction avec le système client : Détection automatiques des propriétés du système
- Interaction avec le réseau : Détection automatique du changement du réseau.

i. Choix d'architecture

On avait à choisir entre plusieurs solutions possibles :

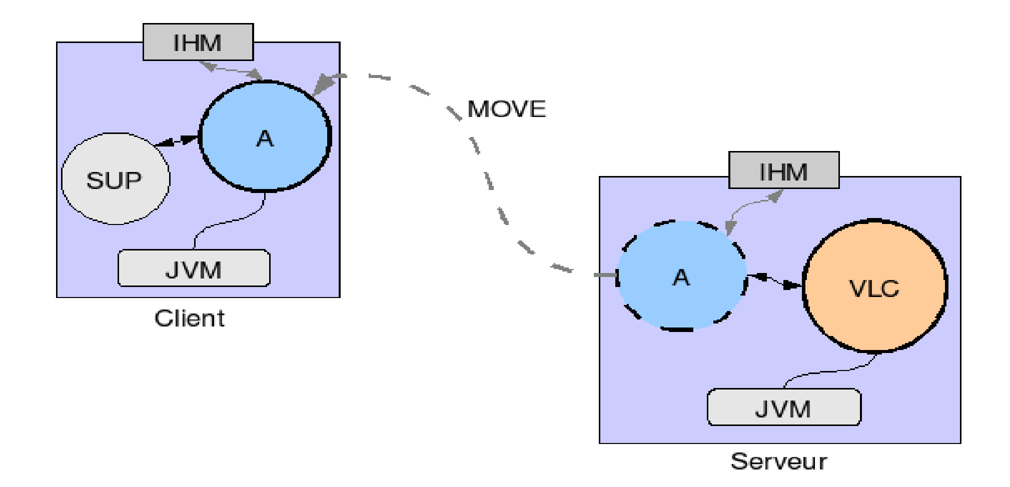
a) 1ère solution



L'agent superviseur est préalablement installé sur le client. Du côté serveur il y a un agent serveur et un agent annuaire qui contient les références des différents agents superviseurs. Là où il se déplace l'agent client récupère la référence de l'agent superviseur à partir de l'annuaire.

Cette solution est complexe à déployer mais elle garantie une certaine rapidité d'exécution.

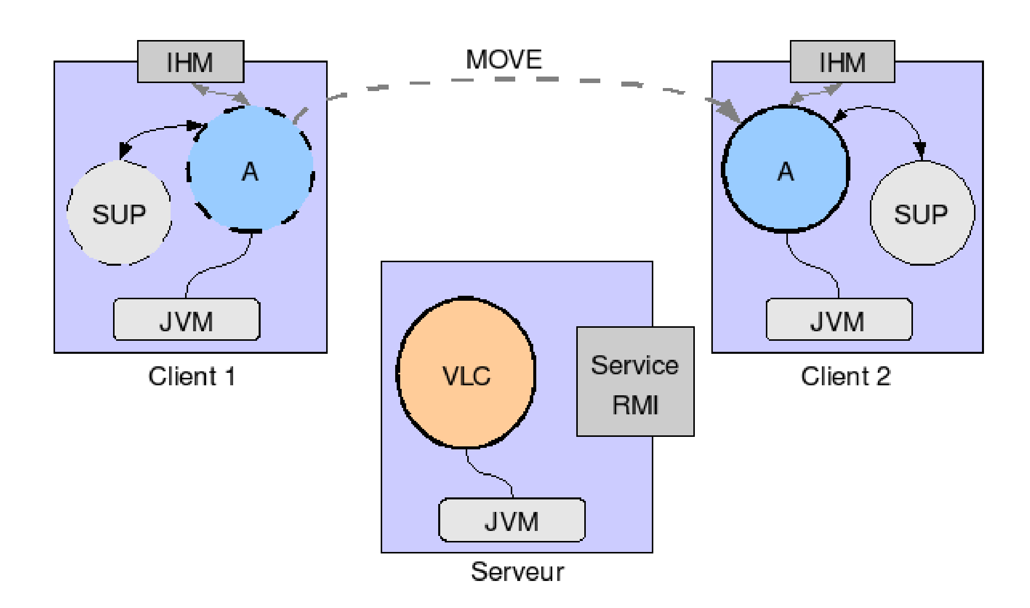
b) 2ème solution



L'agent serveur crée lui même l'agent client relié à l'IHM au niveau du serveur. Puis l'agent client crée le superviseur là où il se déplace.

Cette solution est plus simple à réaliser que la première mais elle n'est pas pratique vu qu'elle suppose que l'utilisateur ait le contrôle du serveur.

c) 3ème solution



Dans cette solution l'agent client n'est pas créé par l'agent serveur. Il se trouve dès le départ là où l'application va être exécutée pour la première fois. Il crée lui même l'agent superviseur à chaque fois où il se déplace. D'un autre côté il récupère la référence de l'agent serveur par un service RMI.

Cette solution est plus simple à déployer que la première et plus pratique que la deuxième. Par contre elle pourrait être assez lourde, surtout si le superviseur prend beaucoup de temps à s'exécuter à chaque fois.

d) Conclusion

On a décodé de commencer par implémenter la troisième solution puis voir si on pourrait évoluer vers la première.

ii. Conception de l'IHM du client

L'IHM est une fenêtre contenant deux objets : un objet JVLC affichant le flux vidéo et un panneau contenant les éléments graphiques suivants :

- Liste ChannelsList : sélectionner la chaîne télé.
- Liste DebitsList : sélectionner le débit du flux.
- Bouton Size : ouvre une fenêtre de dialogue pour saisir la taille de l'image en hauteur et en largeur.
- Bouton Scale : ouvre une fenêtre de dialogue pour saisir la valeur de l'échelle.
- Bouton Start : démarrer/redémarrer le flux vidéo.
- Bouton Stop : arrêter le flux vidéo.
- Bouton Move : ouvre une fenêtre de dialogue pour saisir l'adresse IP du client destination.

B. Codage

i. Côté serveur

La partie serveur est composé de deux agents. Un agent «controleurVLC» qui s'occupe exclusivement de la gestion du serveur VLC et un agent «serveur» qui s'occupe de gérer la réception des messages envoyés par le client.

a) Comportement de l’agent controleurVLC

Voici l'interface de l'agent controleurVLC:

```
public interface ControleurVLC extends ActorProfile, BehaviorProfile {
	public void startVLC(Hashtable<String,String> a);
	public void stopVLC();
}
```

- A la réception du message JAMstartVLC l’agent controleurVLC démarre ou de relance le serveur VLC. Elle prend en paramètre une table de hachage qui contient tous les éléments nécessaires au lancement de VLC. Cette table de hachage est parcourue à l'aide de la fonction updateContent. Elle utilise une API Shell afin d'exécuter et d'interpréter les arguments passés en ligne de commande à VLC. Nous utilisons l'interpréteur de commande sh pour entre autres récupérer le pid du processus lancé et récupérer l'adresse du binaire vlc et vérifier sa présence sur le système.
- A la réception du message JAMstopVLC le contrôleur arrête les processus en cours. Il se charge de tuer le processus externe ainsi que le fils créé lors du lancement de l'interpréteur de commande.

b) Comportement de l’agent serveur

Voici l'interface de l'agent serveur:

```
public interface Serveur extends ActorProfile, BehaviorProfile {
	public void startServeur();
	public void stopServeur();
	public void updateDebit(String d);
	public void updateSizeImage(String x, String y);
	public void updateScale(String s);
	public void updateClientIP(String m);
	public void updateChaine(String c);
}
```

- A la réception du message JAMstartServeur il vérifie tout d'abord si le client a bien communiqué son adresse IP puis envoie un message à l'agent controleurVLC pour démarrer le serveur VLC.
- A la réception du message JAMstopServeur il envoie simplement un message à l'agent controleurVLC pour arrêter le serveur VLC.

- A la réception des messages JAMupdateDebit, JAMupdateSizeImage, JAMupdateScale, JAMupdateClientIP ou JAMupdateChaine, il envoie un message à l'agent controleurVLC pour mettre à jour le champ correspondant dela table de hachage de la classe.
- L'agent serveur héberge également un service RMI. Celui-ci permet de passer la référence de l'agent serveur au client et ainsi de permettre à ces deux entités de communiquer.

ii. Côté client

La partie client est composée de deux agents. Un agent «superviseur» qui supervise l'exécution de l'agent sur le système hôte et un agent «client» qui s'interface avec l'IHM et communique avec l'agent serveur.

a) Comportement de l'agent superviseur

Voici l'interface de l'agent serveur:

```
public interface Superviseur extends BehaviorProfile, ActorProfile {

      public void findProperties();
      public void memoryControl();
}
```

- A la réception du message JAMfindProperties il récupère les propriétés du système hôte.
- A la réception du message JAMmemoryControl il calcule périodiquement la mémoire libre.

b) Comportement de l'agent client

Voici l'interface de l'agent client :

```
public interface Client extends BehaviorProfile, ActorProfile {

    public void initialiser(Actor superviseur);
    public void afficher();
    public void sendChannel(String m);
    public void sendDebit(String m);
    public void sendSize(String x, String y);
    public void sendScale(String m);
    public void start();
    public void stop();
    public void sendIP(String m);
}
```

- Au démarrage de l'agent client, il crée l'agent superviseur et lui envoie le message JAMinitialiser. A travers un service RMI il retouve la référence du serveur puis affiche l'IHM.
- Le client reçoit le message JAMinitialiser de la part du programme principal du Skeleton2. Le client envoie alors les messages JAMfindProperties et JAMmemoryControl à l'agent superviseur.
- A la réception du message JAMafficherle client appelle la méthode show() de l'IHM.
- Le client reçoit tous les autres messages (démarrage, arrêt et modification du flux vidéo) de la part de l'IHM. Il renvoie alors les messages correspondants à l'agent serveur.

C. Tests et validation

Les tests unitaires que nous avons effectués sont les suivants :

i. Tests validés du fonctionnement du côté serveur

- Lancement manuel du serveurVLC et récupération du flux en local et en distant.
- Lancement du serveur VLC par l'agent serveur.

ii. Tests du fonctionnement du Côté client

a) *Tests validés*

- Fonctionnement de l'agent superviseur.
- Affichage de l'IHM par l'agent client.
- Affichage d'une vidéo locale et d'un flux HTTP par une application basée sur JavaVLC.
- Récupération du flux provenant de l'agent serveur VLC et son affichage par l'application distante basée sur JavaVLC.

b) *Tests non validés*

- Affichage d'un flux RTP par une application basée sur JMF.

iii. Tests de communication entre les agents

a) *Tests validés*

- Envoi des messages de l'agent client vers l'agent serveur, situés sur la même machine.

b) *Tests non validés*

- Récupération du flux provenant de l'agent serveur VLC et son affichage par l'agent client local basé sur JavaVLC.
- Récupération de la référence RMI de l'agent serveur par l'agent client distant, et par conséquent l'envoi des messages de l'agent client vers l'agent serveur distant.

V. Conclusion

Ce projet nous a permis de découvrir une nouvelle approche de gestion de la mobilité par la technologie des agents mobiles. Nous avons dû choisir une architecture qui nous semblait la plus adaptée, mais nous nous sommes heurtés à des problèmes liés au déploiement. Ce serait intéressant d'avoir un middleware qui gère le déploiement de façon à le rendre complètement transparent au programmeur.

Une évolution possible de l'application serait de rajouter à l'agent superviseur une fonction de lancement automatique des machines virtuelles au démarrage, ainsi qu'une fonction de supervision de la charge réseau basée sur un service RMI.

Une autre perspective serait de trouver une solution d'affichage du flux vidéo qui soit stable et fonctionnelle quelque soit le système client.

VI. Annexes

A. Format des comptes rendus des réunions

Projet TNT

Compte Rendu de la Réunion n°1 tenue le 31octobre 2007 à l'INT, Evry

Responsables document :	Ines CHAABOUNI et Loïc CHALAYER	Etudiants ASR 2007/2008

Adresse de la réunion :	Telecom INT 9, rue Charles Fourier, 91000 EVRY, France
Contact :	Ines CHAABOUNI Phone : +33 (0) 622 24 00 84 ines.chaabouni@telecomint.eu Loïc CHALAYER Phone : +33 (0) 671 39 13 90 loic.chalayer@telecomint.eu

Participants

Nom	Entreprise / Organisation	Rôle	Présence
Sébastien LERICHE	TELECOM INT	Encadrant du Projet TNT	Oui
Ines CHAABOUNI	TELECOM INT	Etudiante ASR 2007/2008	Oui
Loïc CHALAYER	TELECOM INT	Etudiant ASR 2007/2008	Oui

Prochaine Réunion

Date	Lieu	Nature de la réunion	Organisée par
08/11/07	Telecom INT, EVRY	Réunion Hebdomadaire du Projet	Ines CHAABOUNI et Loïc CHALAYER

Compte Rendu

Ordre du jour de la prochaine réunion